LBNE DAQ/Run Control Interface Specification

Kurt Biery & John Freeman (FNAL)
Erik Blaufuss & John Jacobsen (UMD)
Tim Nicholl's (Sussex?)
Giles Barr (Oxford)
… *(your name here)*

This is the interface layer between Run Control and ArtDAQ. After some discussions, the ArtDAQ and Run control system experts determined that an intermediate interface layer between the two systems would be the best design for the 35ton DAQ system. This layer is mostly driven by the ArtDAQ architecture, where several concurrent subcomponents are needed to readout individual data sources, all determined by the requested system configuration. There is likely start-up-order dependencies among the DAQ subcomponents that are required. This "middle layer" allows:

- Run Control to be somewhat ignorant of the details of a selected run configuration (number and location of board readers, etc)
- The DAQ team to have a location to cleanly map between Run control commands and configurations for the entire DAQ system and the commands and configurations portions needed by any one DAQ subcomponent, including the number and location of each subcomponent.

It is expected that DAQ subcomponents should still report fine-level monitoring information to the Run Control system as they start up, configure, run and terminate. This will allow for centralized monitoring by DAQ experts of DAQ subcomponent states.

Open questions / issues are in red.

1. Name: needs a pithy name? For now we call it daqiface [feel free to rename/search/replace].
2. Definitions:
    a. Run Control is also known as lbnerc. Its command-line interface is lbnecmd. The main control program is known as lbnecontrol.
3. External interface (to Run Control)
    a. Control Interface:
        i. State transition requests are made by components via XML-RPC
        ii. daqiface should listen at some port for XML-RPC commands. Optionally, it can announce it's location on the network by sending the following ZeroMQ message to lbnecontrol on the control server at port 5000:

```
{"type": "control",
 "name": "artdaq",
 "synchronous": True,
 "host": <host>,
 "port": <port>}
```

        Where <host> and <port> are the appropriate host (string; name or IP) and port name (int). Announcing in this way eliminates the need for the operator to explicitly "control" the component.
        iii. RPC commands:
            1. daqiface should implement/register the following XML-RPC functions:
                a. `state_change(name, requested_state, *args)`:
                Introduce state change to the DAQ system.
                "name" will be "artdaq" (or whatever is sent in 3.a.ii). "requested_state" will be

be one of “stop”, “start”, “recover”, “pause”, ”resume”.

i. *args is an optional set of key-value pairs of arguments and values appropriate to the state transition. At the moment, only “starting” has arguments, and they are:

```
{"run_number": <number>,
 "run_config": <config_name> <config_id>}
```

If the state transition isn’t allowed, e.g. “running”->”recovering”, can the C++ code in daqiface raise an exception here or should this return an error?

b. `state(name)`:

Return the overall state of the DAQ.
“name” will again be “artdaq”.
Return one of “stopping”, “stopped”, “pausing”, “paused”, “error”, “recovering”, “starting”, “running”.

b. Monitoring Interface

i. lbnecontrol accepts messages sent in JSON format via ZeroMQ to port 5000. In addition to the message type defined in 3.a.ii, quantities can be transmitted to lbnecontrol in the following format:

```
{"type": "moni",
 "service": "artdaq",
 "t": "2013-11-05 15:06:12",
 "varname": "event_count",
 "value": 12987115}
```

Note that structured values are OK:

```
{"type": "moni",
 "service": "artdaq",
 "t": "2013-11-05 15:13:22",
 "varname": "active_subcomponents",
 "value": ["event_builder_1": {"state": "running",
                               "events_processed": 139578},
           ...]}
```

ii. When daqiface’s state changes, it should report the new state to run control, e.g.,

```
{"type": "moni",
 "service": "artdaq",
 "t": "2013-11-05 15:06:12",
 "varname": "state",
 "value": "running"}
```

iii. When a subrun starts or stops, daqiface should report the time, run number, and subrun:

```
{"type": "moni",
 "service": "artdaq",
 "t": "2013-11-05 15:06:12",
 "varname": "rundata",
 "value": {"runnumber", 135987,
           "subrun", 2,
           "tstart": "2013-11-05 15:06:10.12389713"}}
```

(We may wish to add information to this message)

c. Logging Interface

i. Log messages (to be displayed to operators and/or archived) can also be sent to run control the same way. In this case "varname" should be "log" and "value" should be the desired string. Example:

```
{"type": "moni",
```

```
"service": "artdaq",
"t": "2013-11-05 15:19:01",
"varname": "log",
"value": "Ack!  Something really bad happened, I don't know what..."}
```


d. Alert Interface
    i. [An alert interface for run control is planned but not yet implemented. The alerts may send messages to email inboxes and/or mobile phones, and will be displayed on the Run Control Web pages.]

4. Internal interactions with ArtDAQ:
    a. [04-Dec, KAB: unfortunately, I'm not completely clear on which state transitions we will make visible to the operators. In some of our conversations, this has been suggested to be as simple as "start" and "stop". A more full-featured set might include "configure", "start", "pause", "resume", "stop", "shutdown". It would be great to settle on these soon. To get started on this section, I will presume the simple model (but truthfully, my preference would be something more verbose).]
    b. when the DAQinterface layer receives the "start" command, it will
        i. start all of the artdaq processes
        ii. tell the ConfigurationManager to generate the desired configuration files
        iii. send the "init" command to each of the artdaq processes, in the appropriate order
        iv. send the "start" command to each of the artdaq processes, in the appropriate order
        v. if RC sends a "state" request at any time during this process, the DAQinterface layer will return a state of "starting"
        vi. if an error is encountered in any of the individual steps, what should the DAQinterface layer do? One proposal would be for it to simply stop and report a state of "error" to any "state" requests from RC. RunControl could then send a "recover" request at some convenient time.
    c. when the DAQinterface layer receives the Stop command, it will
        i. send the "stop" command to each of the artdaq processes, in the appropriate order
        ii. send the "shutdown" command to each of the artdaq processes, in the appropriate order
        iii. if RC sends a "state" request at any time during this process, the DAQinterface layer will return a state of "stopping"
        iv. if an error is encountered in any of the individual steps, what should the DAQinterface layer do? One proposal would be for it to simply stop and report a state of "error" to any "state" requests from RC. RunControl could then send a "Recover" request at some convenient time.
    d. It would be great to discuss and define the choices that operators are allowed to make through the RC GUI when configuring the system and starting runs. This should be folded into other parts of the document, but I'll make some notes here until we have a chance to discuss this.
        i. At a minimum, the user will need to choose a named configuration.
            1. John and Kurt have tentatively discussed having RunControl query the ConfigurationManager to obtain the list of available configurations and display those to the user.
        ii. In the simplest model, RC would send the run number, configuration name, and configuration ID in the "start" message.
        iii. A more flexible interface would allow the user to select which parts of the detector to include in the run and the number of event builder nodes to use. These choices would be passed to the DAQInterface layer, and it would start the appropriate number (and type) of BoardReaders and number of EventBuilders based on the user's choice.
    e. Maybe it would be useful to list the options so we can all refer to them in our discussions:
        i. Options for which operations are presented to the user
            1. start, stop, and recover
            2. start, stop, pause, resume, and recover

3. configure, start, stop, pause, resume, recover, and shutdown
4. other option(s)?
5. (Obviously, the trade-offs are simplicity vs. finer-grained control, and possibly less time between runs if no reconfiguration is needed.)

ii. Options for what fraction of the configuration is specified by the user

1. the user picks a single named configuration and that specifies all of the system, hardware, artdaq software, and art software parameters
2. the user specifies the parts of the detector to be included, the number of event builders to run, whether to write the data to disk, and a single named configuration that specifies the remaining parameters (hardware, artdaq software, and art software)
3. other option(s)

5. Project disposition:
   a. daqiface lives in the ArtDAQ software distribution and is built, installed and started (in the program sense, not the state management sense) as part of same.

Please email john@mail.npxdesigns.com if you would like edit access to this document.